プロシージャルゲームコンテンツ制作ブートキャンプ 2022
『グランツーリスモ7』

「レースゲーム用地形のポリゴンリダクションについて」

株式会社　ポリフォニー・ディジタル　齋藤 彰

POLYPHONY™ DIGITAL

自己紹介

株式会社ポリフォニー・デジタル　景観デザインチーム
チーフ　テクニカル　アーティスト
齋藤 彰

経歴：
1995年株式会社ナムコ (現・株式会社バンダイナムコエンターテインメント) 入社
主にアーケードゲーム開発で、テクニカルディレクション、アートディレクション、メカデザインを担当。

2003年株式会社ポリフォニー・デジタル入社
景観デザインチームにて、主にテクニカルディレクション、ツール開発、
世界各地への取材に始まり最終データの調整まで、グランツーリスモシリーズのコース制作のあらゆる過程に携わらせて頂いております。

CEDEC2017「Houdini16にHDAを2つ追加したらビルが量産できた(プロシージャルモデリング事始め)」

CEDEC2018「プロシージャルゲームコンテンツ制作ブートキャンプ Part 1 はじめに　Procedural Hard Surface Design With HOUDINI」

SIGGRAPH2018 ASIA2018 HOUDINI HIVE「Procedural Hard Surface Design」

CEDEC2019「プロシージャルゲームコンテンツ制作ブートキャンプ 2019 Part 1 バリエーション生成」

Houdini HIVE Japan 2020「Procedural Hard Surface Design」

POLYPHONY™ DIGITAL

今回お話しする事

・レースゲームの地形モデルのポリゴンリダクションを、
どのように効果的に行ったか。

・プロシージャルワークフローにおいて、私がとても重要だと思っている、
形状を数学的に分析し操作対象を自動で選択する部分を、
地形のポリゴンリダクションを例に解説します。

・Houdini 19を使用して解説していますが、
他のDCCツールなどでも応用できる内容です。

POLYPHONY™ DIGITAL

前提

・地形生成ワークフローについて。
・グランツーリスモ7での地形データについて。

POLYPHONY™ DIGITAL

POLYPHONY™ DIGITAL

POLYPHONY™ DIGITAL

地形生成ワークフローについて

・グランツーリスモ7 では、Houdiniを使用した自動処理が多く導入されています。

・画面に表示されるポリゴン以外にも、コリジョン等のメタデータの一部も
Houdiniにより自動生成されています。

・コース(レベル)デザインや絵作りのイテレーション効率を上げる為に、
最終工程に近いポリゴンリダクションの自動化をHoudiniで行いました。

POLYPHONY™ DIGITAL

グランツーリスモ7での地形データについて

・地形データは、測量やHoudiniのTerrain等によるシミュレーションで用意しています。
(詳しくは、本日17:30　第8会場『グランツーリスモ7』のコース制作における測量技術と外観デザインの事例)

・グランツーリスモ7では、基本的にサーキットなど限定された空間を走行するゲームデザインの為、オープンワールドゲームで一般的なハイトマップを用いたテッセレーションメッシュではなく、直接ポリゴンによって地形がモデリングされています。

・無駄なポリゴンを削減するための効率の良いポリゴンリダクションは必要不可欠。

POLYPHONY™ DIGITAL

ポリゴンリダクションのポイント

・どこから見るモデルなのか？

・一様にポリゴンを減らすのではなく、

「減らすべき場所」

「元の形状を保持すべき場所」

この2つを分けるのが大切です。

POLYPHONY™ DIGITAL

「減らすべき場所」　「裏面」と「影」

- 「裏面」：視点に対して反対向きのポリゴン
- 「影」：視点から何かに遮蔽されて見えないポリゴン

POLYPHONY™ DIGITAL

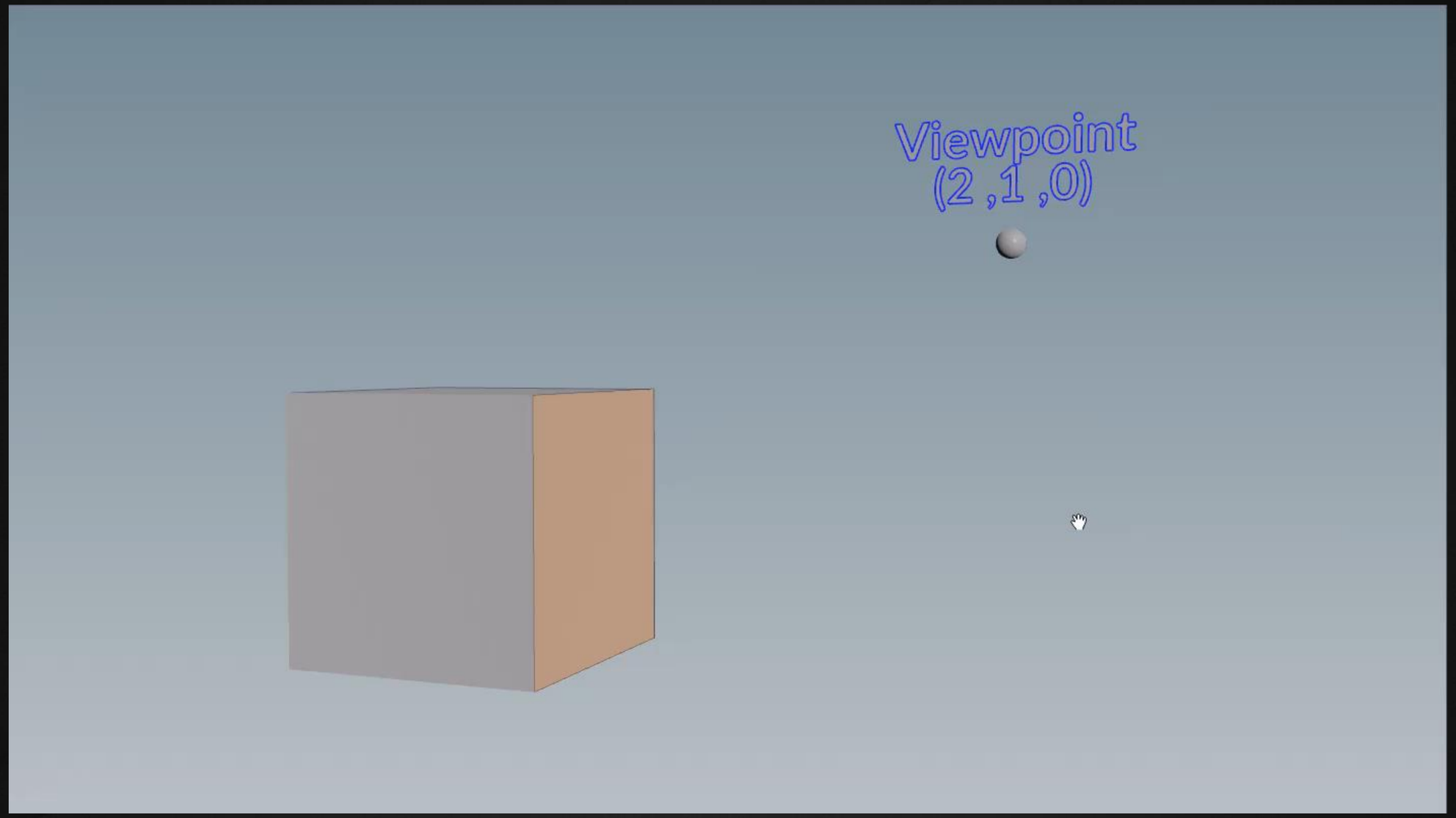
裏面

・内積を用いる事で判断する事が出来ます。

・dot(ポリゴンの法線、ポリゴンから視点への向き）
　結果がプラスなら表面、マイナスなら裏面。

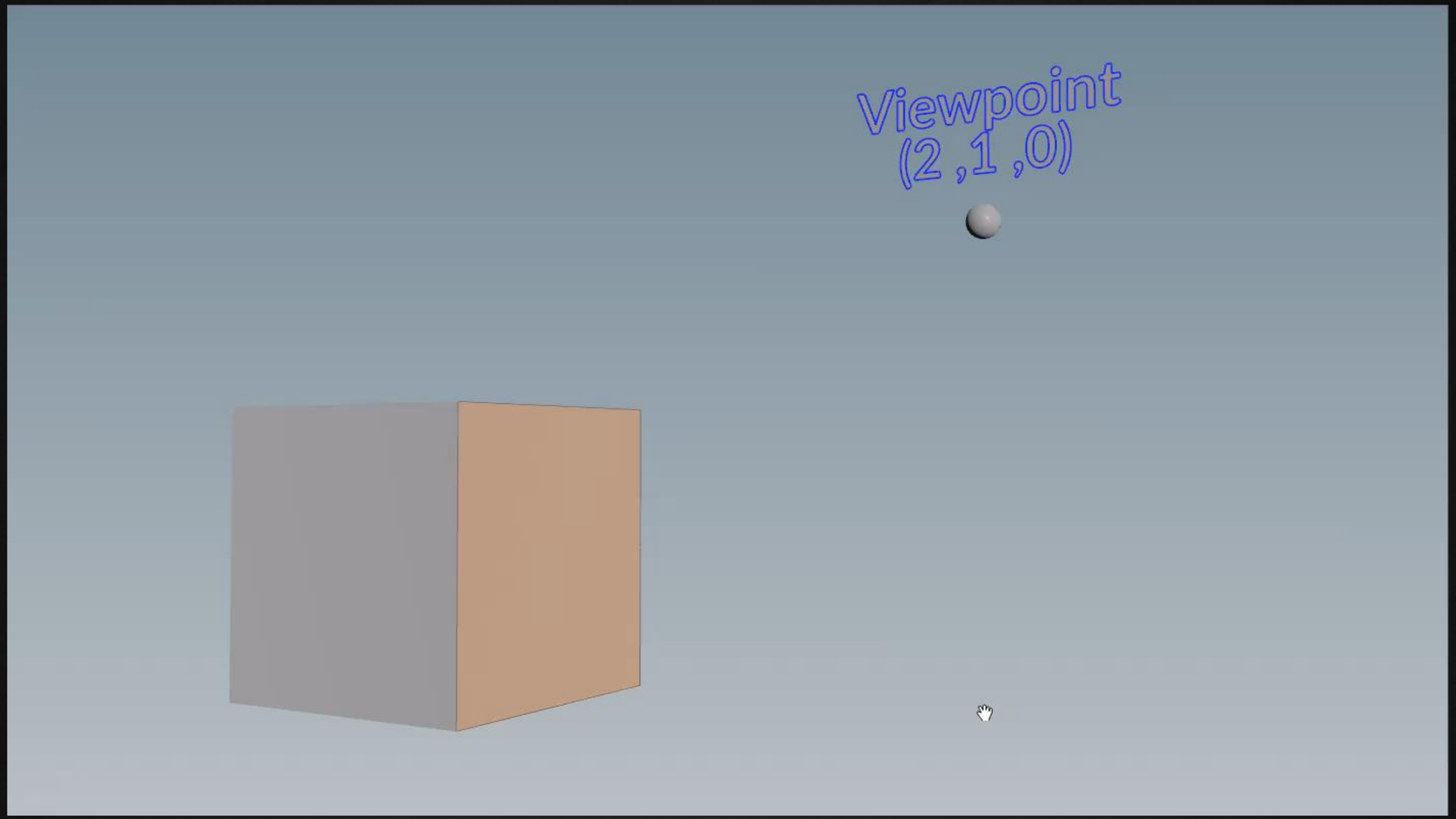
POLYPHONY™ DIGITAL

ポリゴンの向き(法線)を外積で



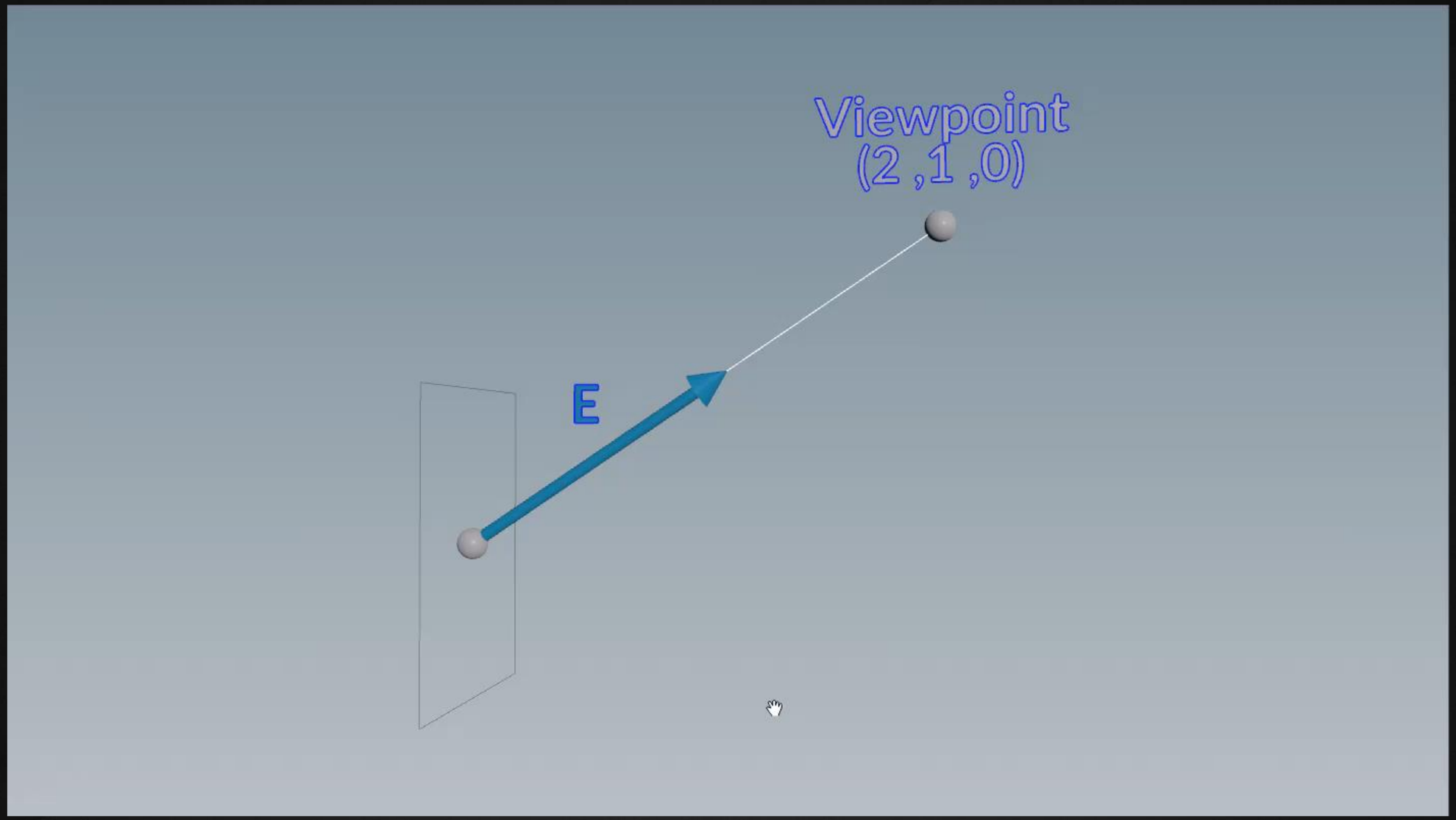
POLYPHONY™ DIGITAL

ポリゴンから視点への向き



POLYPHONY™ DIGITAL

法線と視線への向きの内積



POLYPHONY™ DIGITAL

VEX(Run Over Primitives)で書くと

Parameters | Visualizers

Code | Bindings

Group

Group Type: Guess from Group

Run Over: Primitives

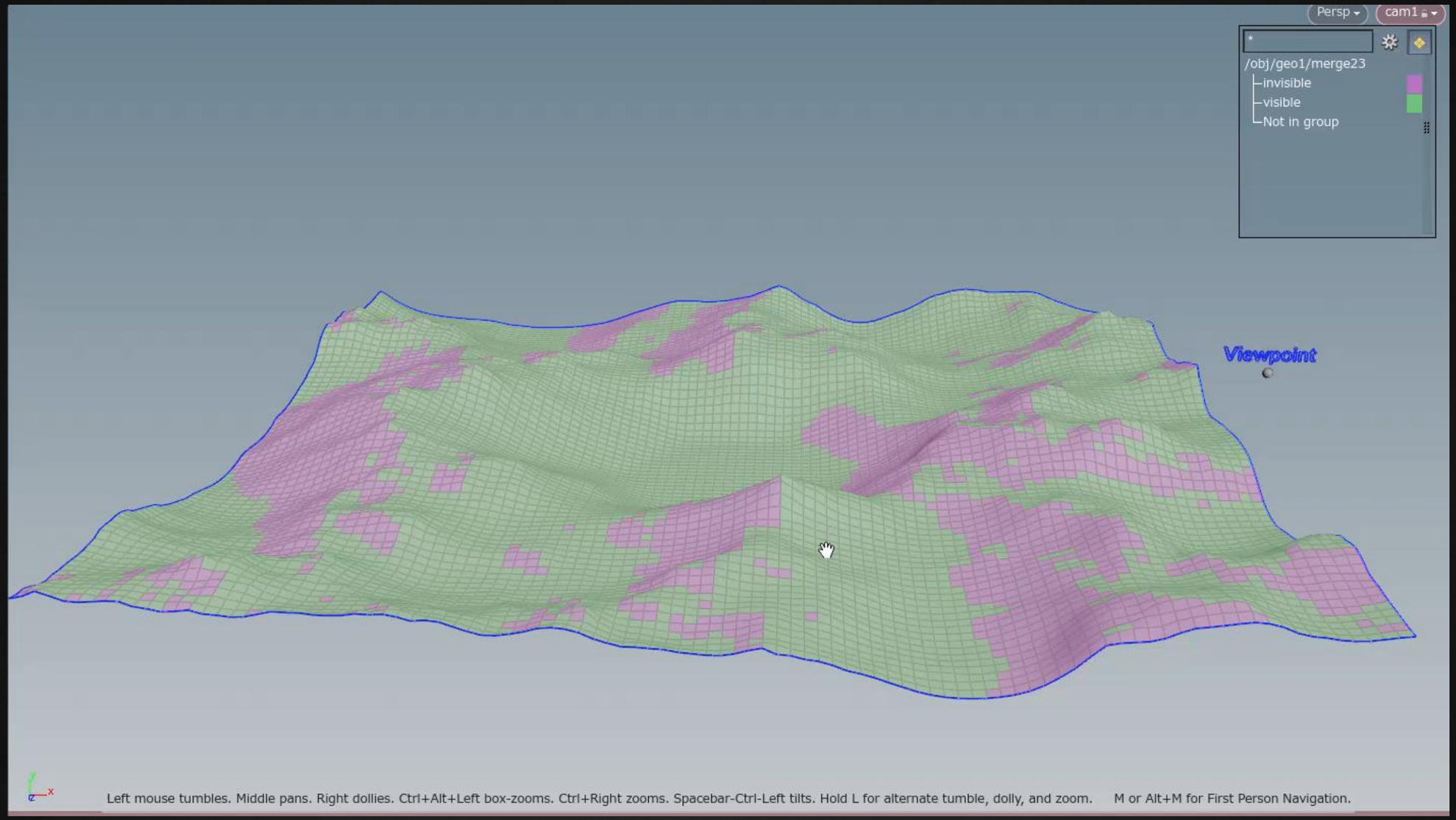
VEXpression

```
1  //input0に対象のメッシュ
2  //input1に視点として設定されたポイント
3  //面の法線は事前にnormal SOP等により@Nに格納されている
4
5  //視点の位置を取得
6  vector viewpoint = point(1,"P",0);
7
8  //ポリゴン中心から視点までの正規化されたベクトルを求める
9  vector v0 = normalize(viewpoint - @P);
10
11 //面の法線と、視点までのベクトルの内積を求める
12 f@facingRate = dot(@N,v0);
13
14
15 if(f@facingRate > 0)
16 {
17     //内積の結果が0より大きければ表面
18     i@group_visible = 1;
19 }else{
20     //内積の結果が0以下であれば裏面
21     i@group_invisible = 1;
22 }
```

Ln 4, Col 1

POLYPHONY™ DIGITAL

裏面検出デモ


Persp
cam1
/obj/geo1/merge23
invisible
visible
Not in group
Viewpoint
Left mouse tumbles. Middle pans. Right dollies. Ctrl+Alt+Left box-zooms. Ctrl+Right zooms. Spacebar-Ctrl-Left tilts. Hold L for alternate tumble, dolly, and zoom.
M or Alt+M for First Person Navigation.

POLYPHONY™ DIGITAL

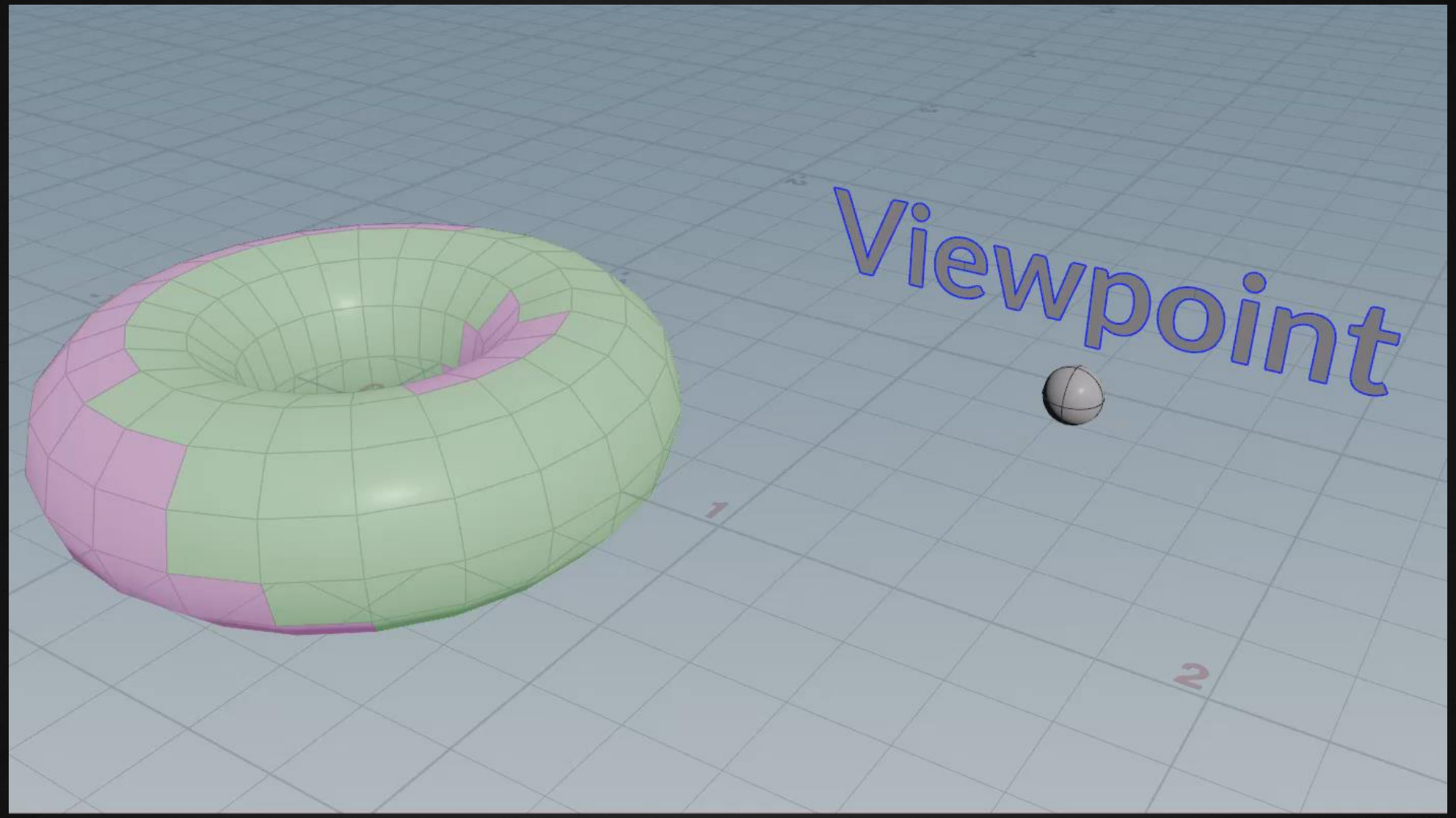
残るは「影」

- 「裏面」：視点に対して反対向きのポリゴン
- 「影」：視点から何かに遮蔽されて見えないポリゴン

「影」は「裏面」ではない「表面」に存在します。

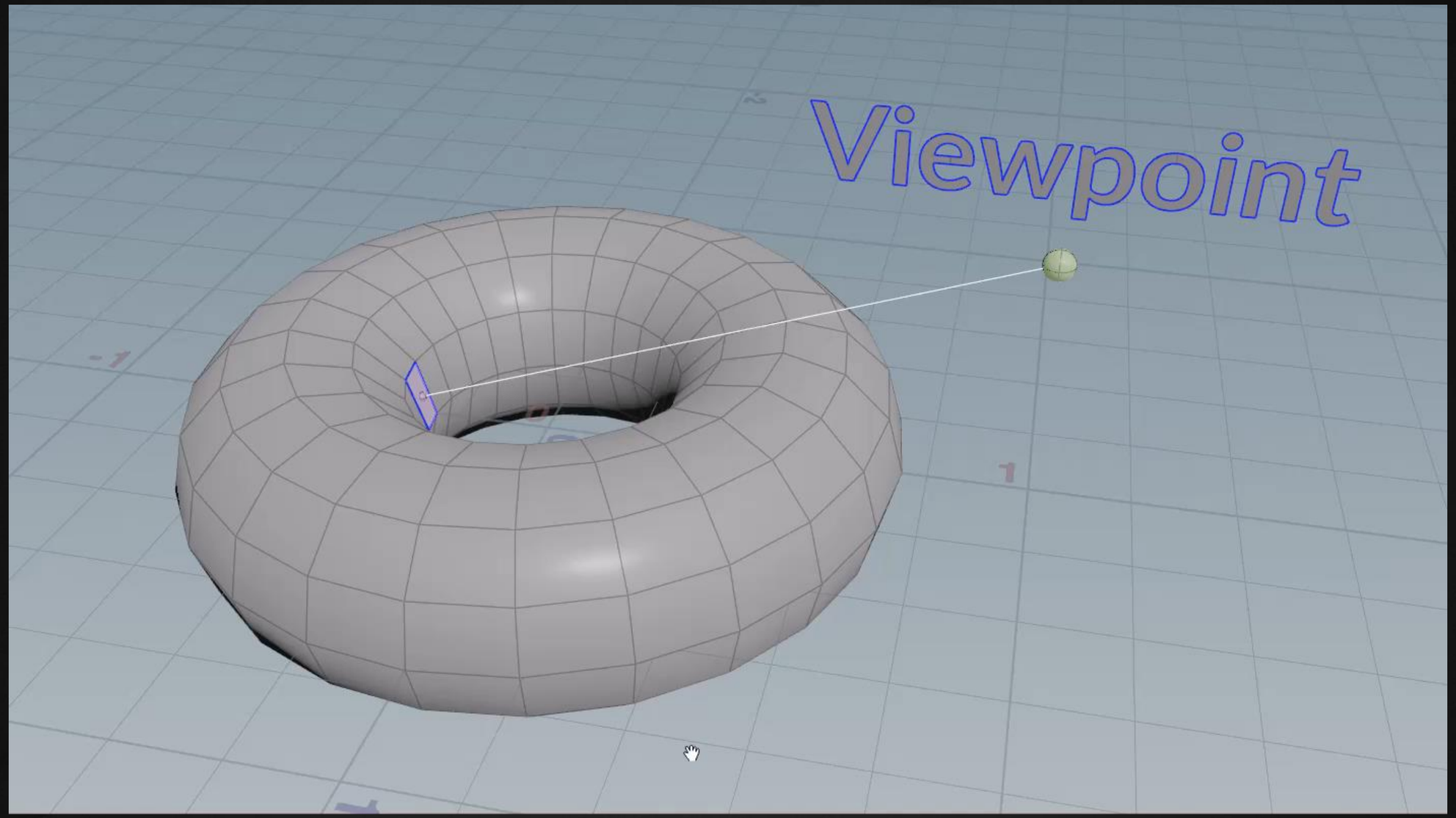
POLYPHONY™ DIGITAL

「影」:視点から何かに遮蔽されて見えないポリゴン



POLYPHONY™ DIGITAL

「影」:視点から何かに遮蔽されて見えないポリゴン



POLYPHONY™ DIGITAL

線分と三角形の交差判定をVEX(RunOver primitives)で書くと

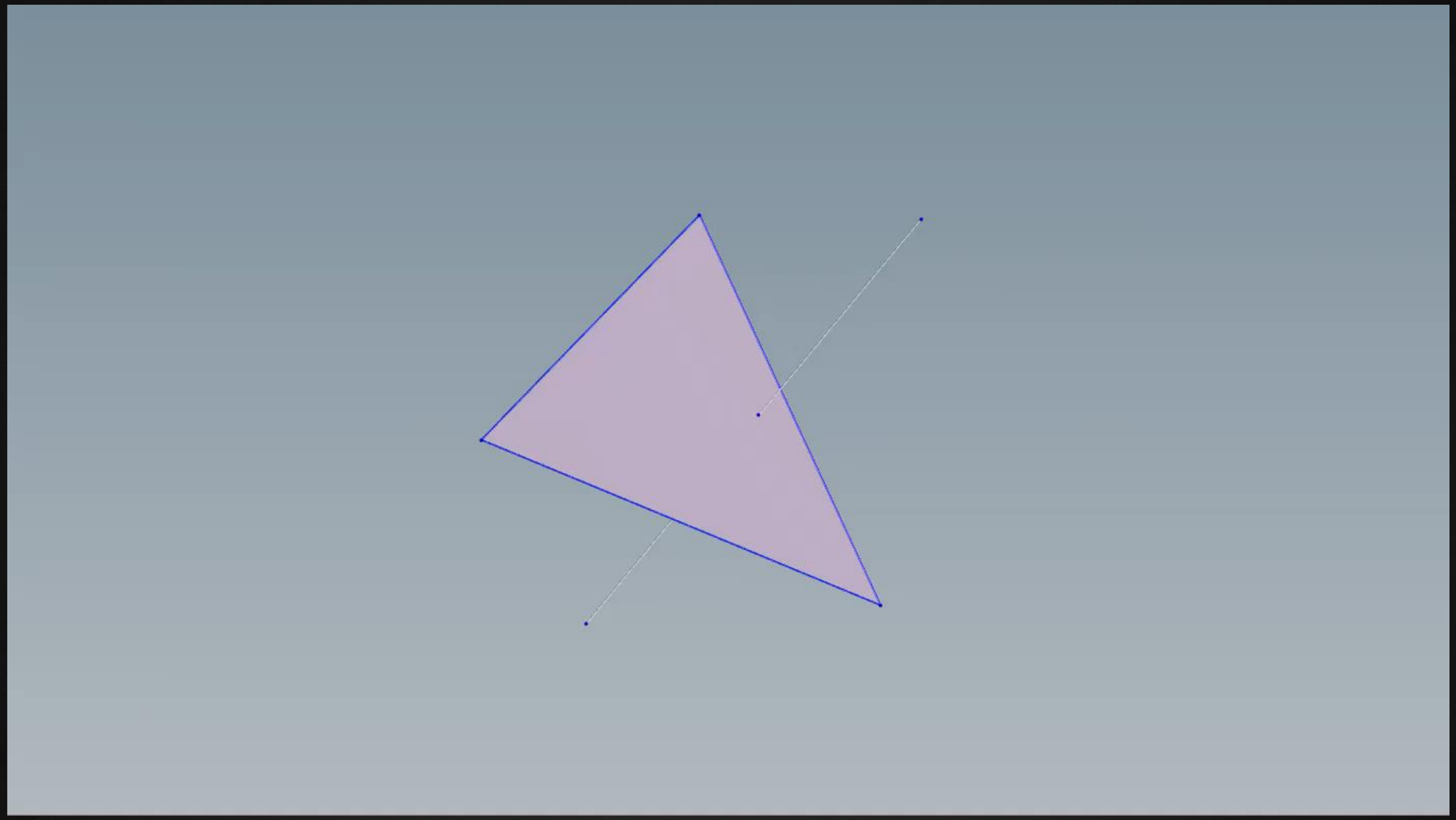
```
//線分の端点の座標
vector l0 = point(1,"P",0);
vector l1 = point(1,"P",1);

//三角形の各頂点の座標
vector p0 = point(0,"P",0);
vector p1 = point(0,"P",1);
vector p2 = point(0,"P",2);

//三角形の中心座標（平面の原点座標）
vector s0 = prim(0,"P",0);

//三角形の法線「外積」を用いて求める
vector n0 = normalize(cross(p2-p0,p1-p0));

//三角形の中心から、各端点へのベクトル
vector v0 = l0 - s0;
vector v1 = l1 - s0;

//平面と線分との交差判定
//平面の原点から各端点のベクトルと平面の法線の「内積」の符号が相反する場合、
//その平面と線分は交差している。
if(dot(n0,v0) * dot(n0,v1)<=0)
{
    //平面と線分が交差していた場合の処理

    //三角形の中心から、各端点へのベクトルと法線の「内積」の絶対値を求める事により、
    //平面から線分の端点までの距離を求める事が出来る。
    float d0 = abs(dot(v0,n0));
    float d1 = abs(dot(v1,n0));
    //求めた距離から内分点の計算に必要な比率を求める。
    float r = d0/(d0+d1);
    //三角形の１点と線分の端点を結ぶベクトルを求める。
    vector v2 = l0 - p0;
    vector v3 = l1 - p0;
    //内分点を求め、平面と線分が交差する座標を求める。
    vector p3 = (1-r)*v2 + r*v3 + p0;

    //交点に頂点を生成
    addpoint(0,p3);

    //交点と三角形の内外判定
    //既存の三角形の辺と、新たに出来た頂点のベクトルの「外積」と、三角形の法線の「内積」を求める。
    float dot0 = dot(n0,cross(p3 - p0,p1 - p0));
    float dot1 = dot(n0,cross(p3 - p1,p2 - p1));
    float dot2 = dot(n0,cross(p3 - p2,p0 - p2));
    //全ての内積の結果が正数の場合。交点は三角形の内側にある。
    if(dot0>0 && dot1>0 && dot2>0)
    {
        //交点が内側にあった場合グループを設定
        i@group_hit = 1;
    }
}
```

- 三角形の法線を求める
- 三角形が存在する平面と線分の交差判定
- 平面から線分の各頂点までの距離を求める
- 平面と線分の交点を求める(内分点)
- 三角形と交点の内外判定

注意:安定した平面のため入力を3角形としています。
input0 に三角形１つ、input1に線分が接続されています。

POLYPHONY™ DIGITAL

衝突判定



POLYPHONY™ DIGITAL

衝突判定をVEX(RunOver primitives)で書くと

```
1  //視点位置
2  vector viewpoint = point(1,"P",0);
3
4  //ポリゴン中心から視点位置へのベクトル（光線の方向と距離）
5  vector v0 = viewpoint - @P;
6
7  //衝突した位置とUVを格納する変数(今回は使用していない)
8  vector hitPos,hitUv;
9
10 //光線の原点　自分自身と接触しないために、少しだけ法線方向に浮かせます。
11 vector origin = @P + @N * 0.0001 ;
12
13 //intersect()関数
14 //光線がポリゴンに交差した場合は、そのポリゴン番号
15 //交差しなかった場合は-1を返します。
16 int hit = intersect(0,origin,v0,hitPos,hitUv);
17
18
19 if(hit != -1)
20 {
21     //intersect()関数の結果が-1以外の場合、そのポリゴンを影とします。
22     i@group_shadow = 1;
23 }
24
```

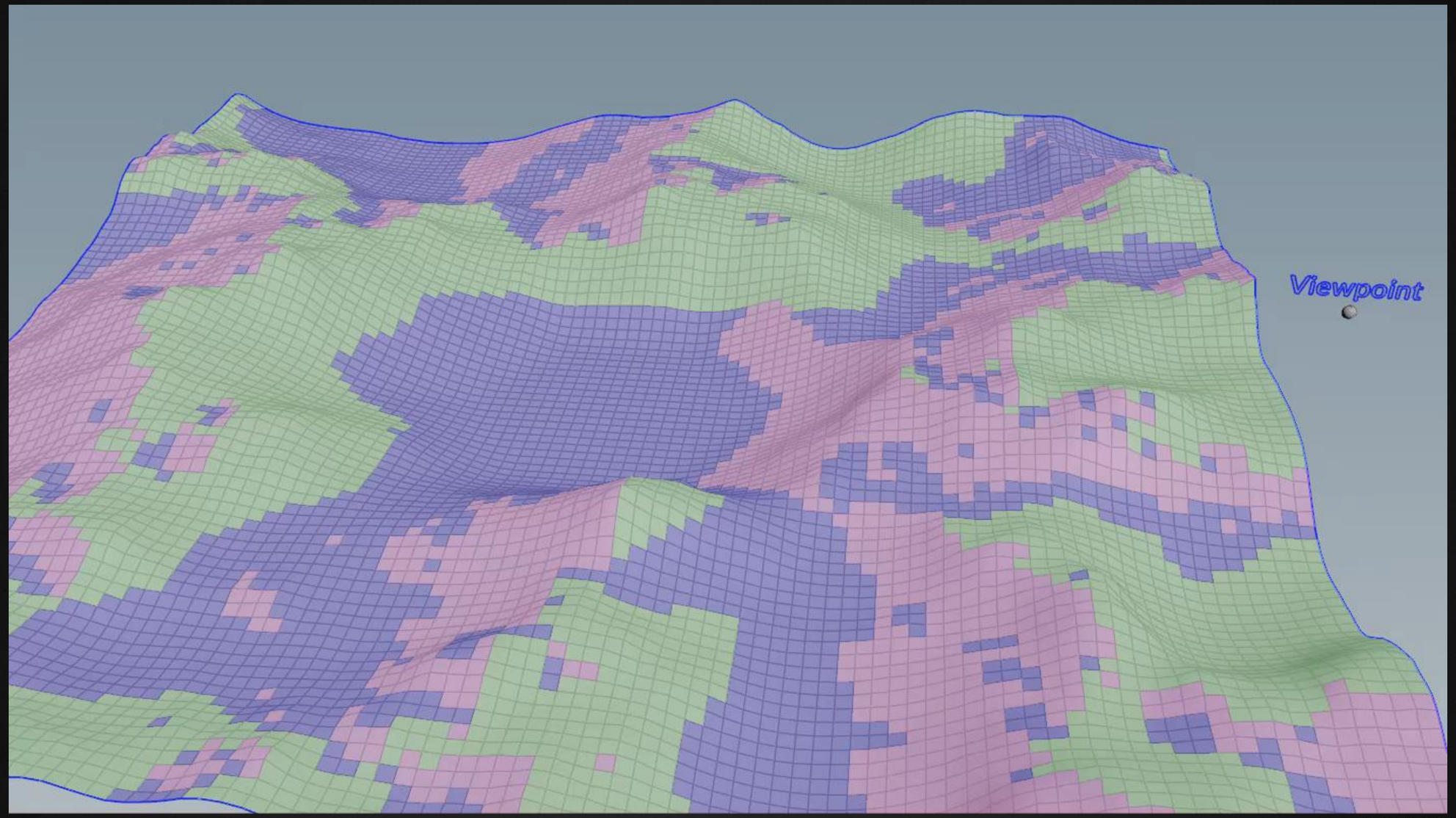
Houdiniには、レンダリング等でのレイトレースや、大量のパーティクルの衝突判定などの為に、様々な交差判定の仕組みが用意されています。

今回は、

Intersect()関数を使用し、交差判定を行っています。

POLYPHONY™ DIGITAL

衝突判定



POLYPHONY™ DIGITAL

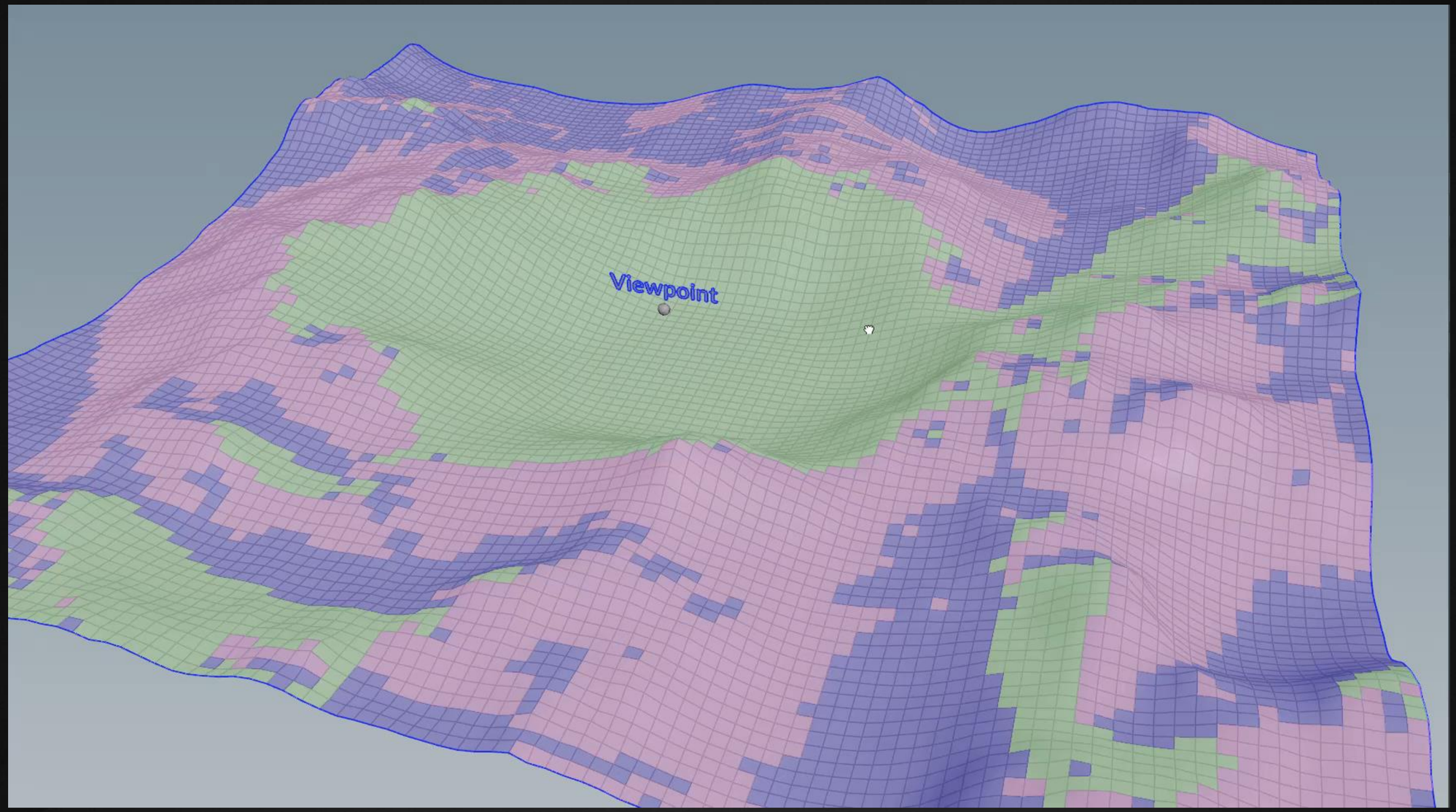
「元の形状を保持すべき場所」

- 「減らすべき場所」
  「裏面」：視点に対して反対向きのポリゴン
  「影」：視点から何かに遮蔽されて見えないポリゴン

- 「元の形状を保持すべき場所」
  「全てのポリゴン」ー「裏面」ー「影」

POLYPHONY™ DIGITAL

「元の形状を保持すべき場所」



POLYPHONY™ DIGITAL

複数の視点

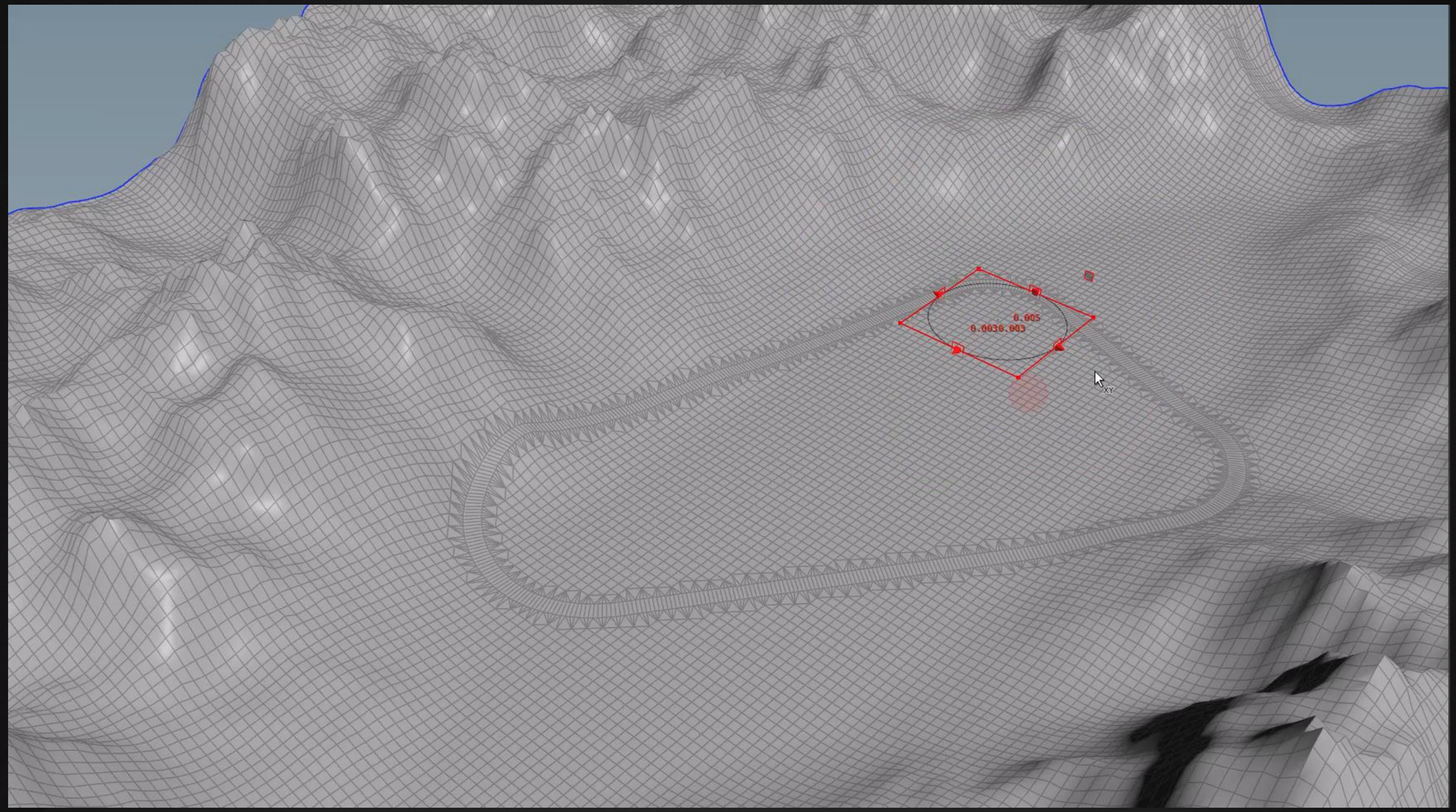
・グランツーリスモ7では、基本的に視点(カメラ)は、自車（ドライバー）視点、追尾視点でゲームが進行するので、自由移動ではあるが、ある程度視点が限定されている。

・そこで、コースのセンターラインを等分割し、想定される視点の高さに上昇させた複数の点を視点とし使用します。

・演出用のカメラがあればその位置も含めます。

POLYPHONY™ DIGITAL

複数の視点



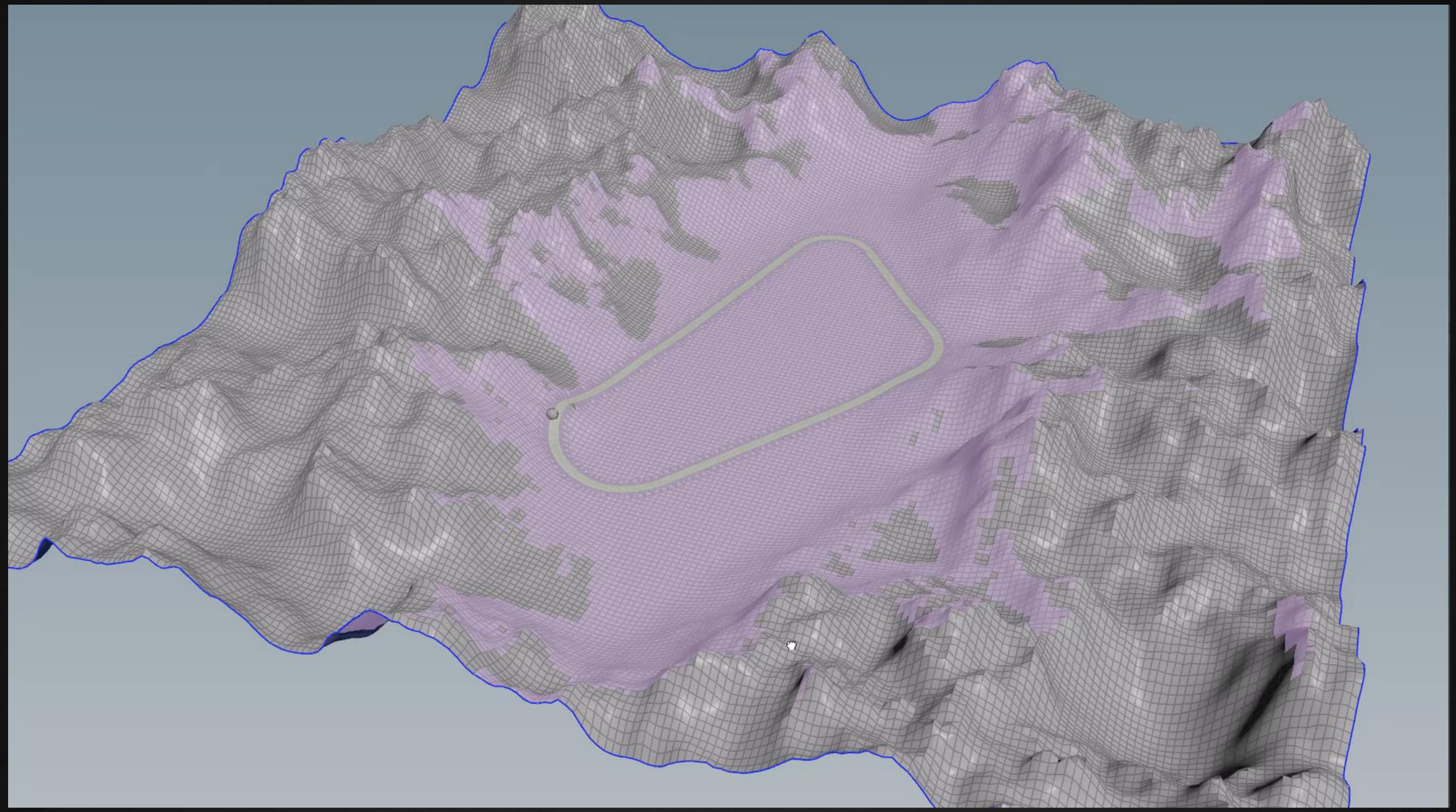
POLYPHONY™ DIGITAL

複数の視点

・各視点、一つ一つから「元の形状を保持すべき場所」を求め、それらを合成することで、全体としての「元の形状を保持すべき場所」を設定します。

POLYPHONY™ DIGITAL

複数の視点の「元の形状を保持すべき場所」で重ね塗り



POLYPHONY™ DIGITAL

リダクション

・HoudiniのポリゴンリダクションPoly Reduce SOPには、
「絶対に削減したくないポイント(エッジ)」 Hard Points(Edges)
「あまり削減したくないポイント(エッジ)」 Soft Points(Edges)
この2つが設定可能

どちらにも指定されていない場所が一番削減されます。

POLYPHONY™ DIGITAL

リダクション

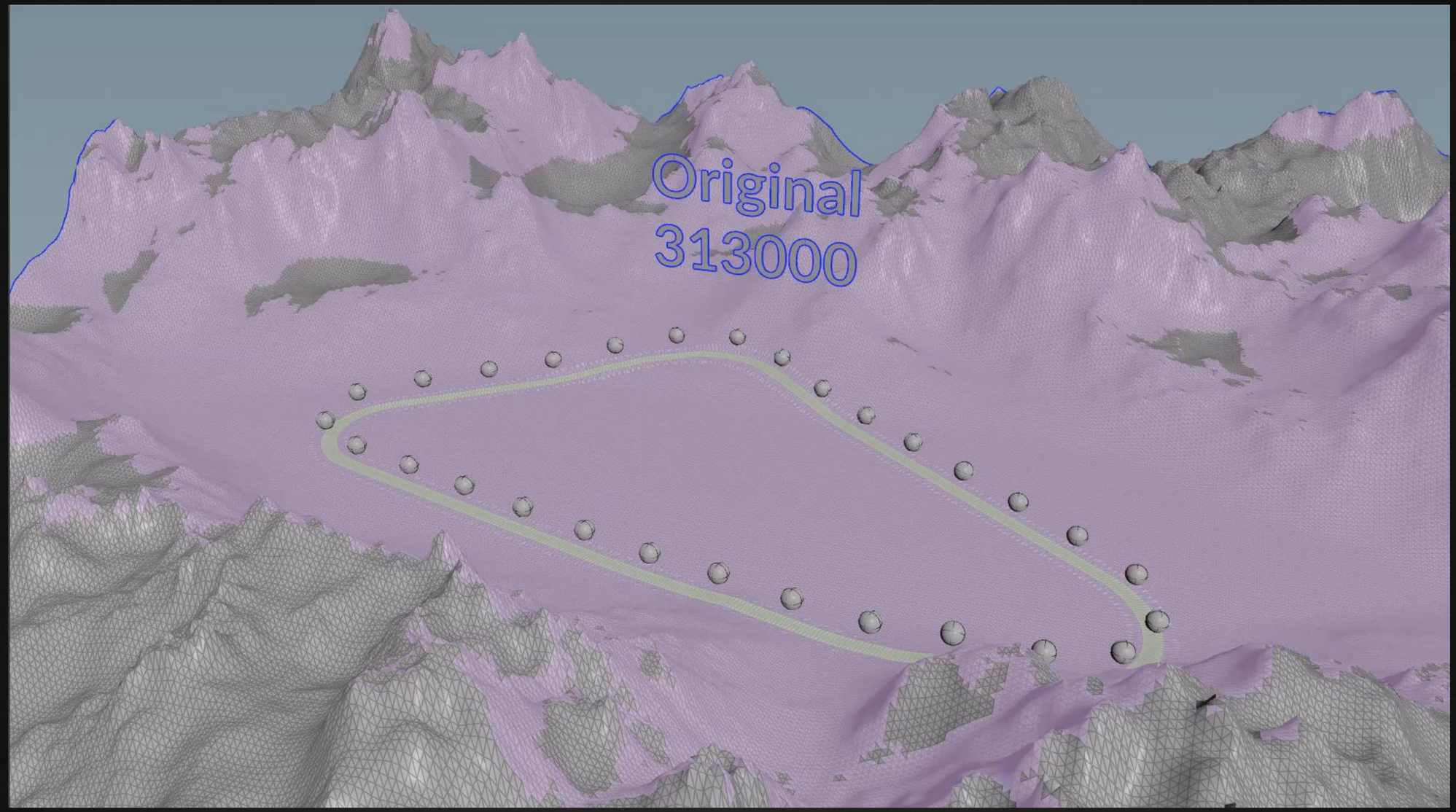
・「絶対に削減したくないポイント(エッジ)」 Hard Points(Edges)
　路面やその近傍を設定。

・「あまり削減したくないポイント(エッジ)」 Soft Points(Edges)
　「元の形状を保持すべき場所」を設定

・何も設定されていない、「裏面」と「影」がより強く削減されます。

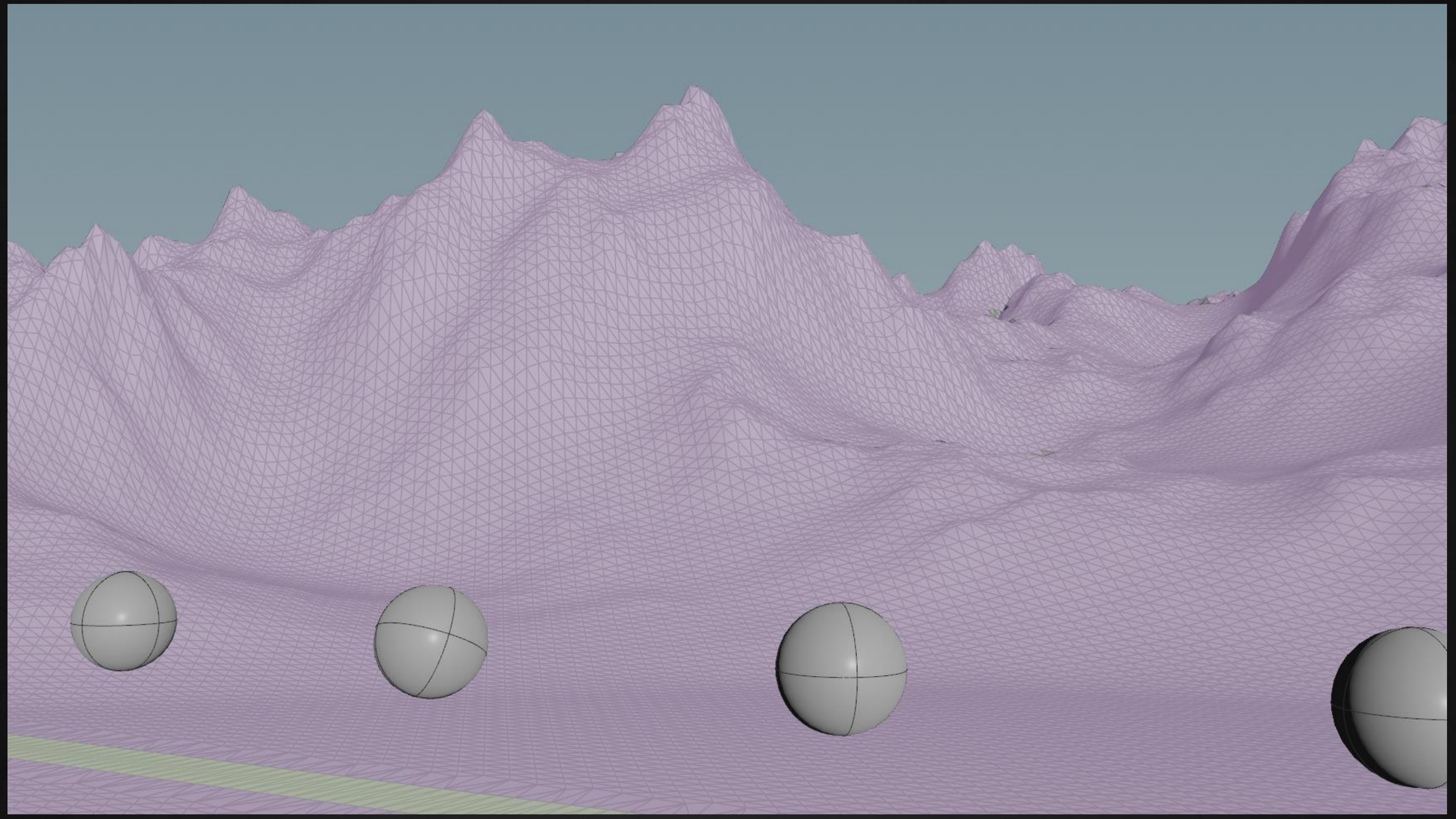
POLYPHONY™ DIGITAL

リダクション結果



POLYPHONY™ DIGITAL

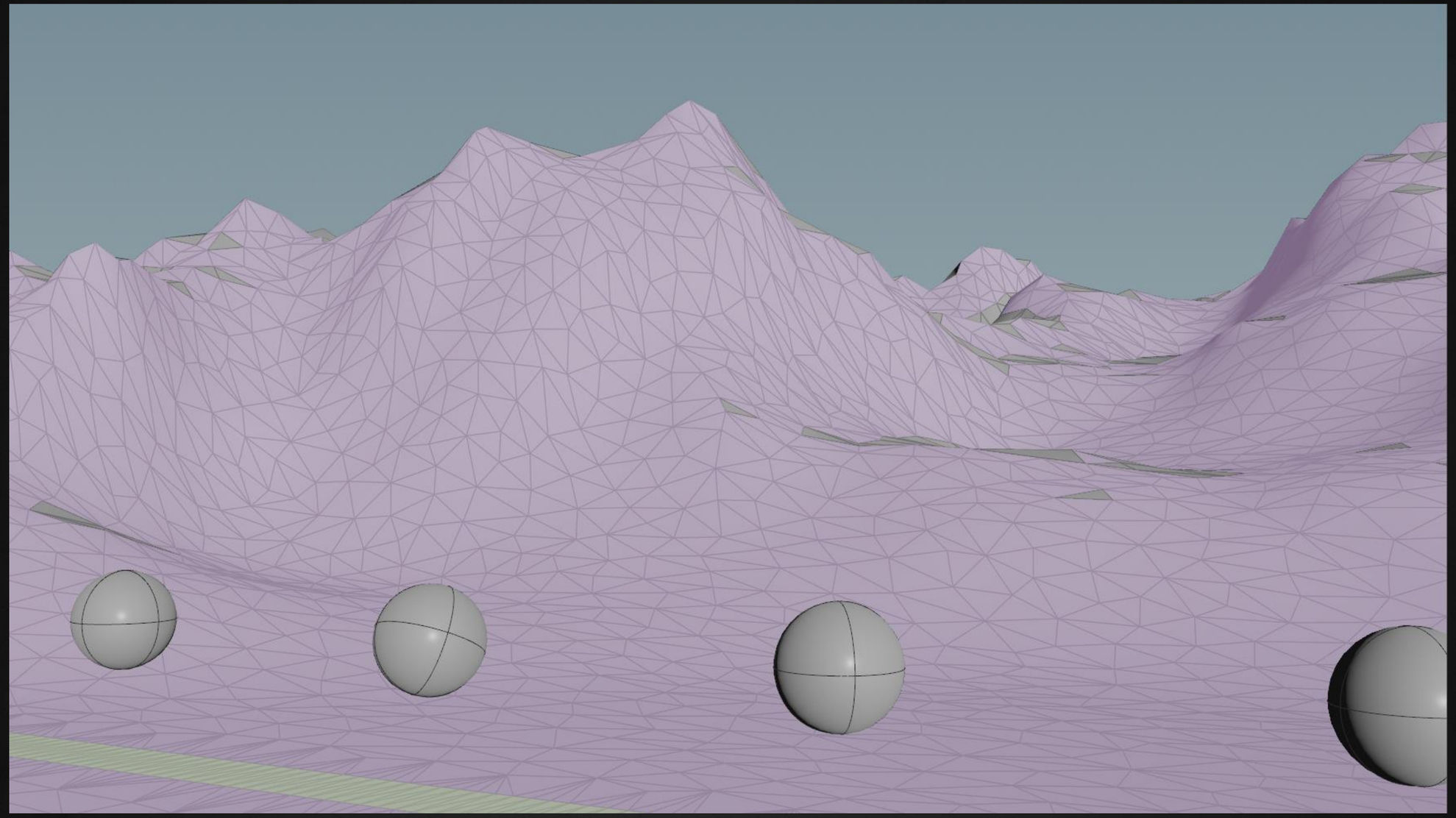
リダクション結果



入力メッシュ

POLYPHONY™ DIGITAL

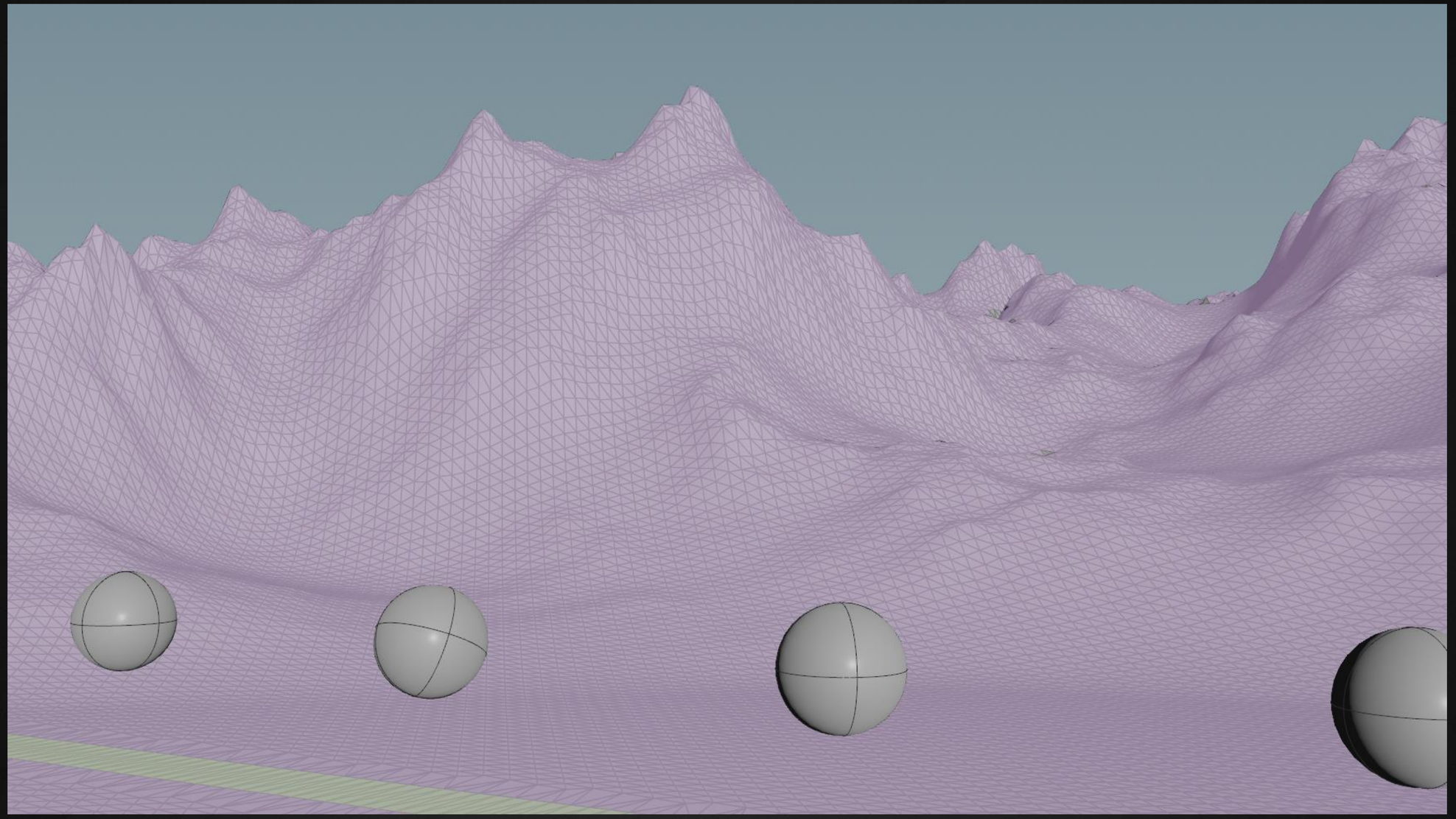
リダクション結果



リダクション後

POLYPHONY™ DIGITAL

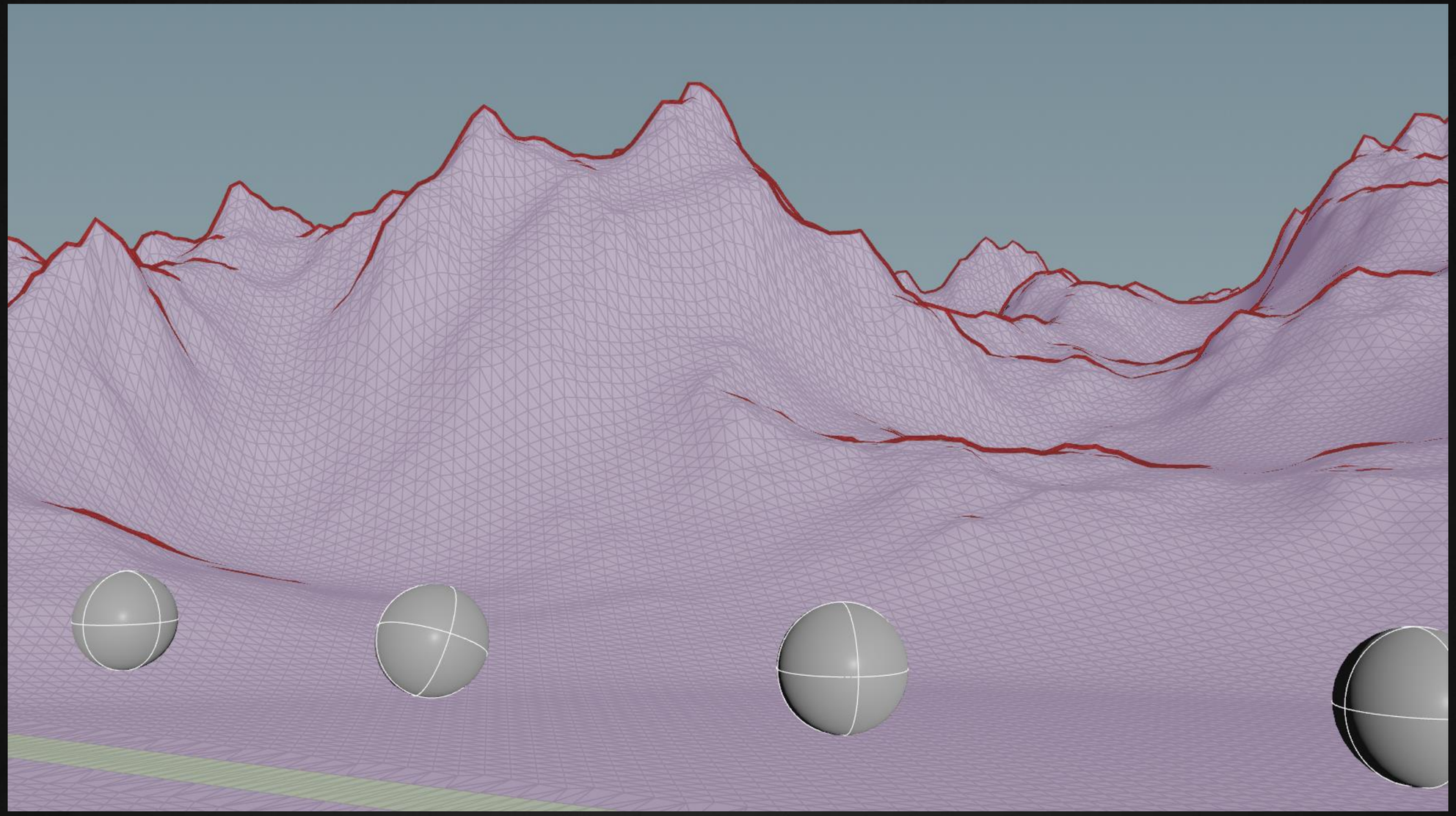
リダクション結果



入力メッシュ

POLYPHONY™ DIGITAL

地形モデルの特徴の一つ「稜線」



POLYPHONY™ DIGITAL

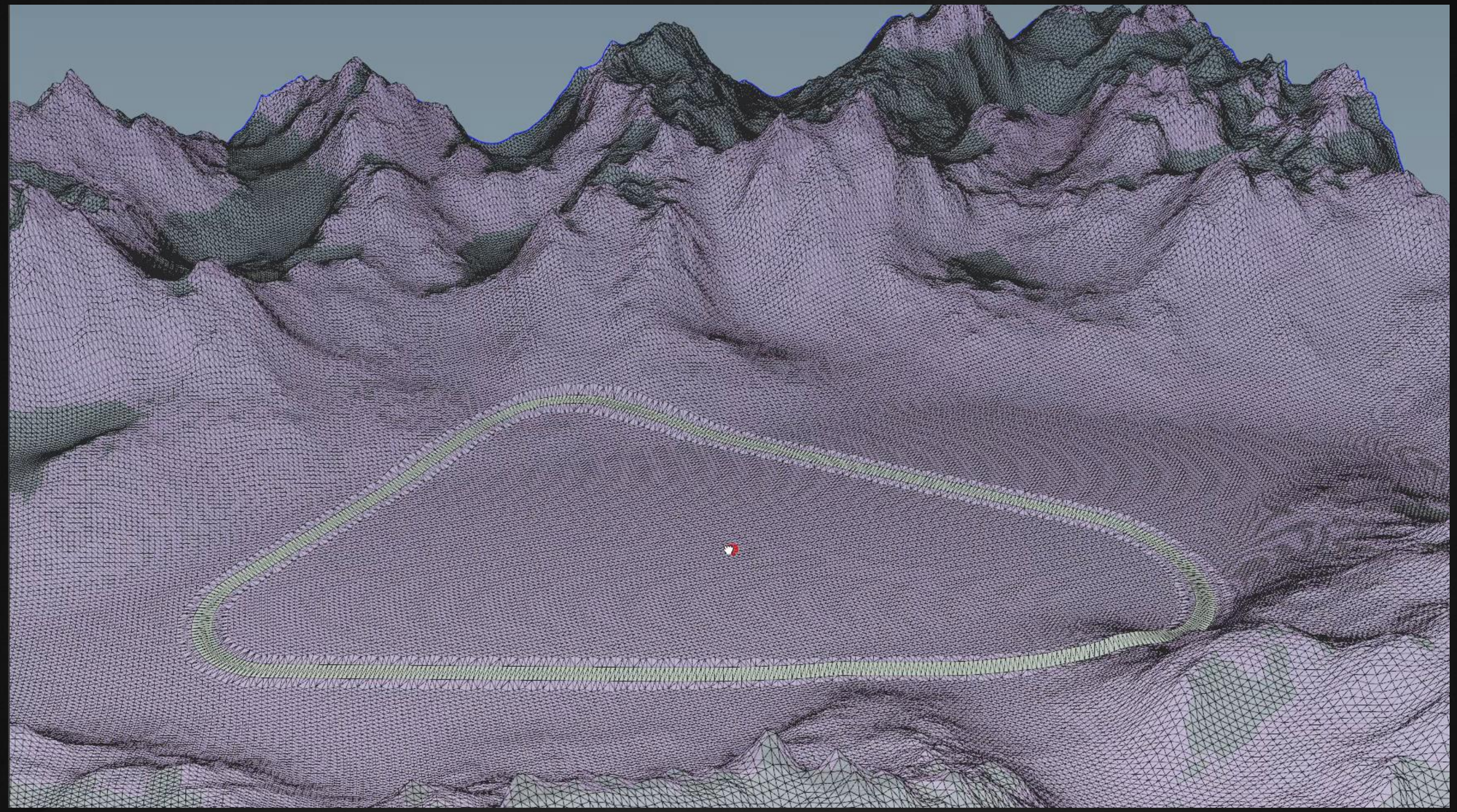
稜線

・山の稜線が削減されすぎているので何とかしたい。

・山の稜線を検出し、
「絶対に削減したくないポイント(エッジ)」 Hard Points(Edges)として扱いたい

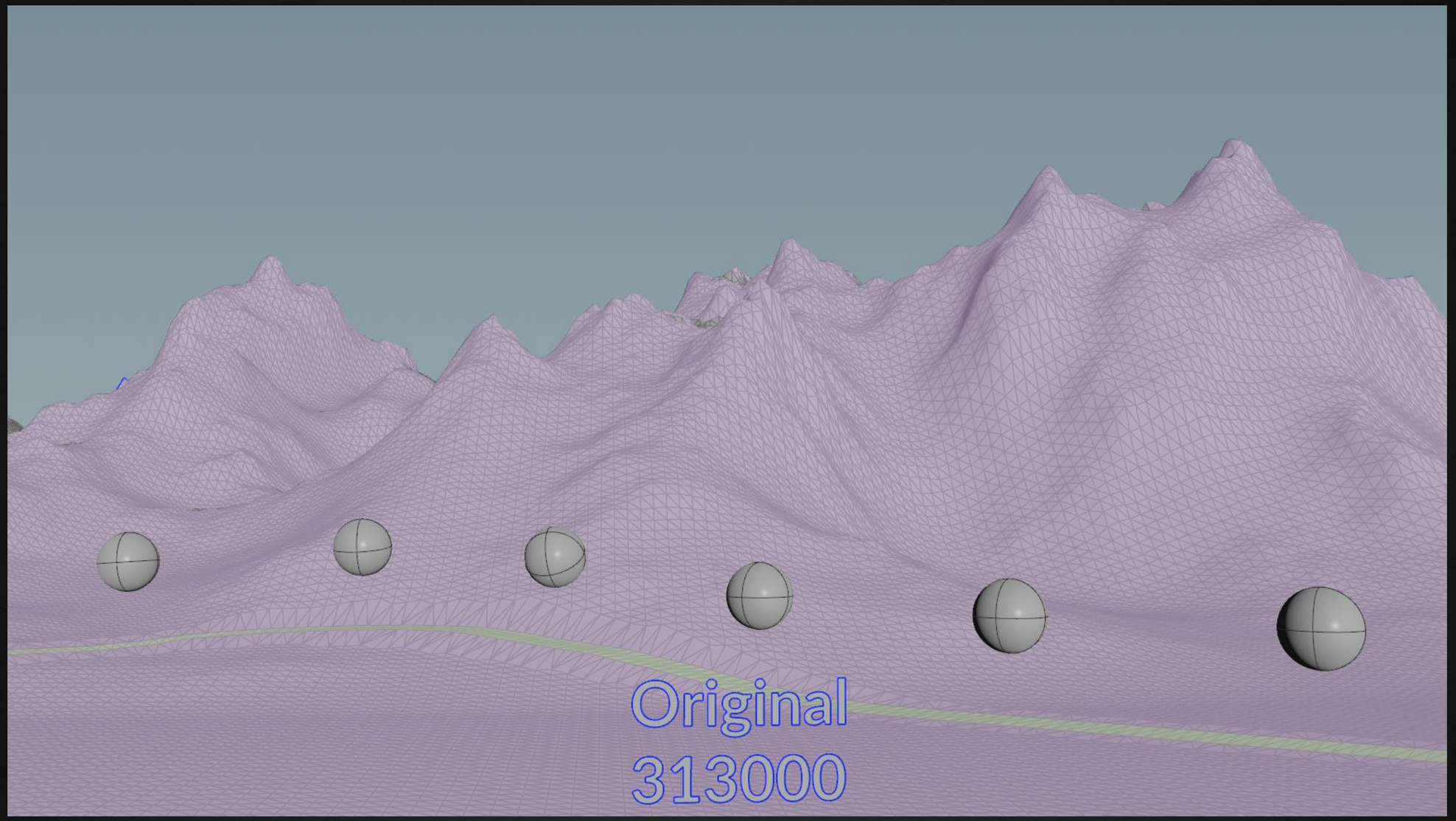
POLYPHONY™ DIGITAL

稜線の検出



POLYPHONY™ DIGITAL

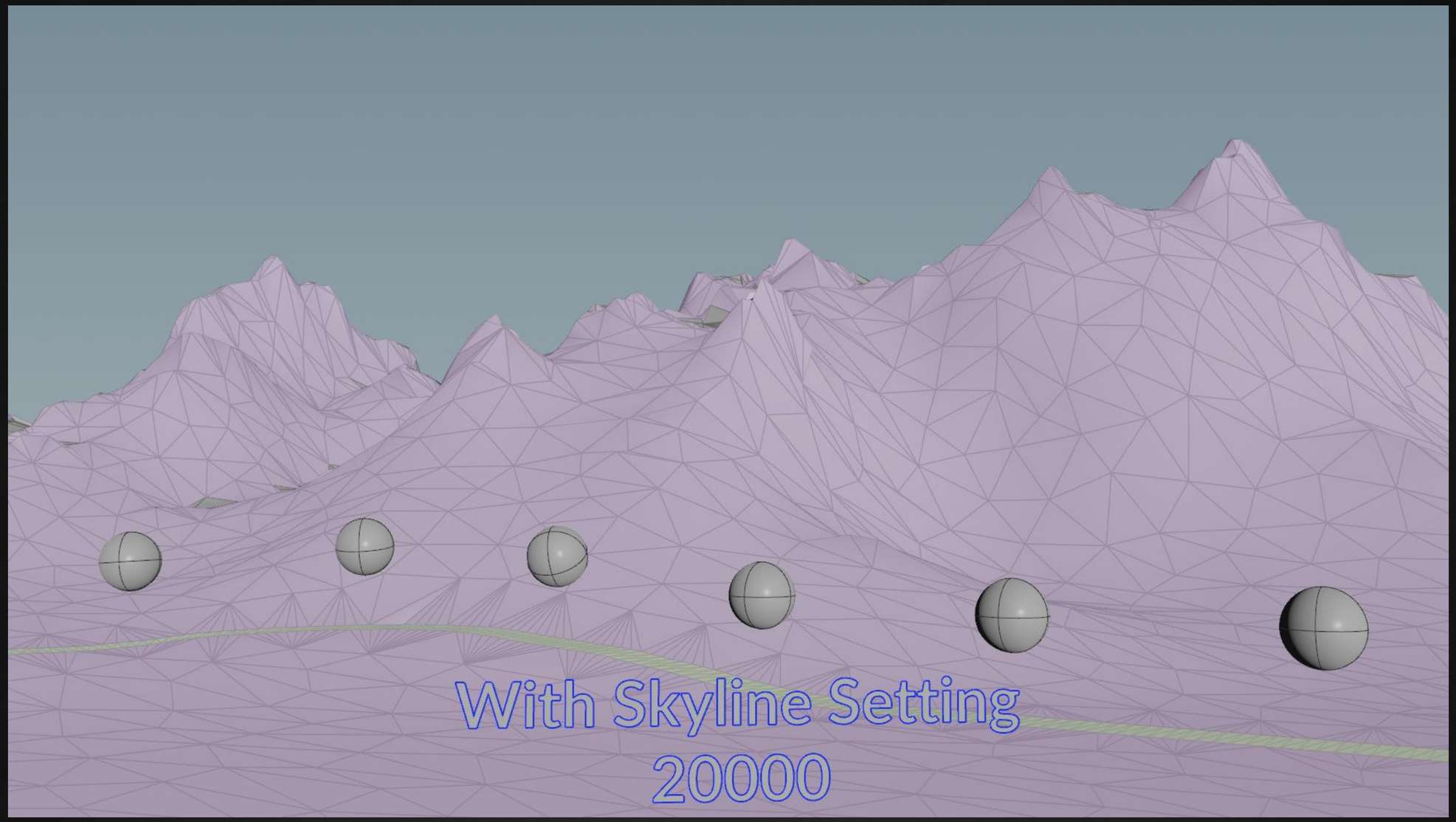
稜線を加味したリダクション



入力メッシュ

POLYPHONY™ DIGITAL

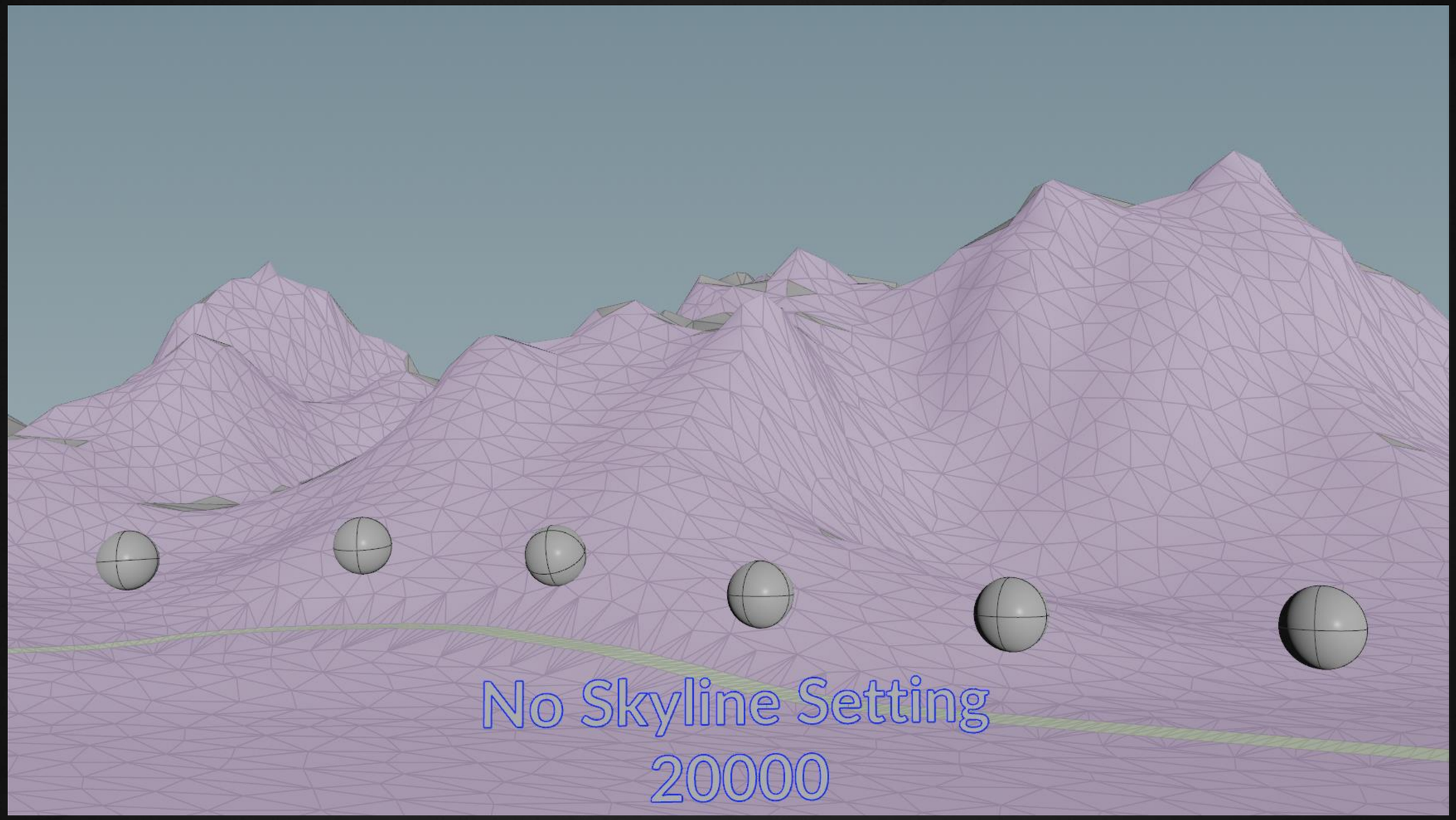
稜線を加味したリダクション



稜線設定有り

POLYPHONY™ DIGITAL

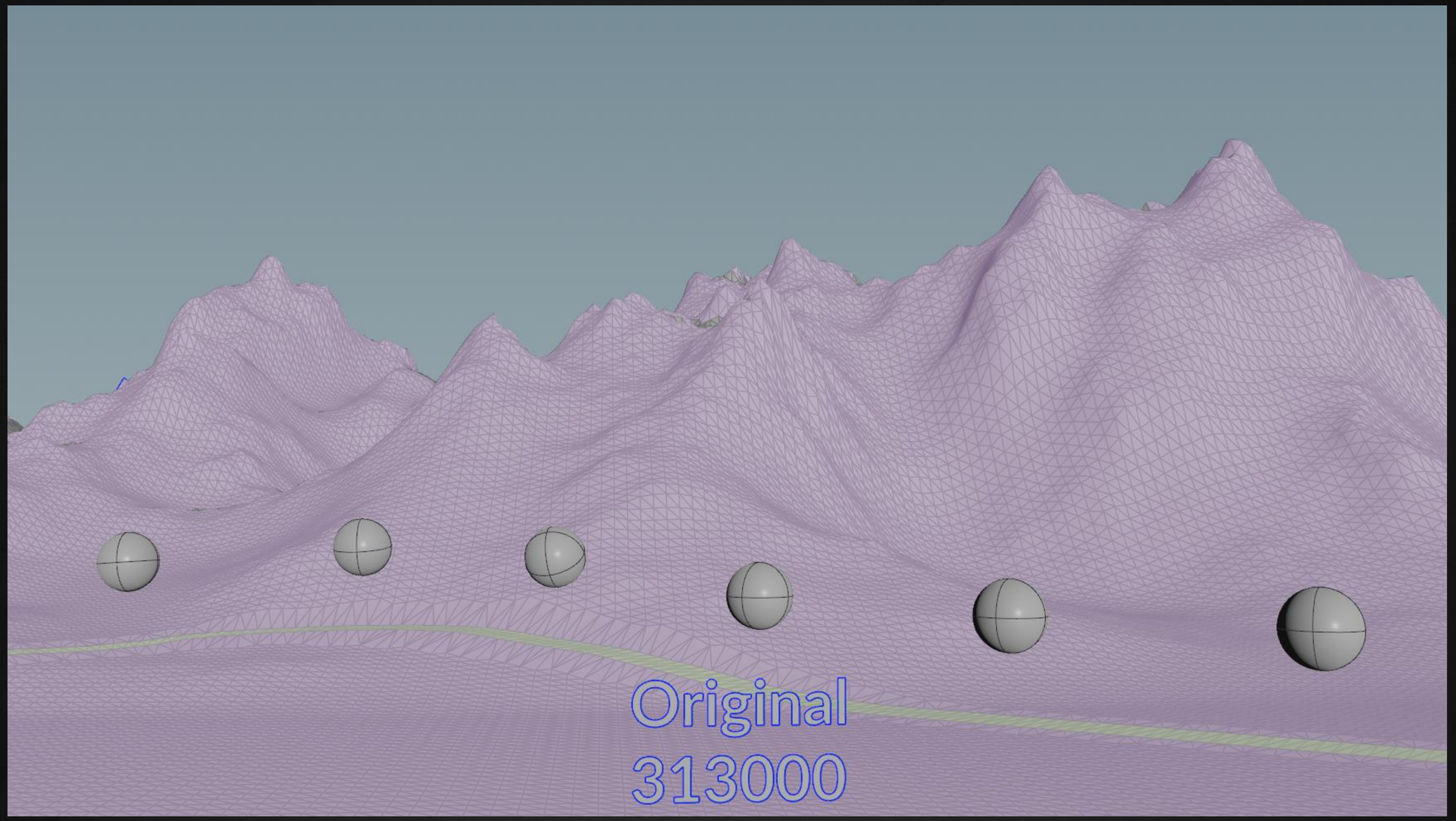
稜線を加味したリダクション



稜線設定無し

POLYPHONY™ DIGITAL

稜線を加味したリダクション



入力メッシュ

POLYPHONY™ DIGITAL

補足情報と、まとめ

・今回は実際の製品で使用した手法を簡略し紹介しました。
ポリゴン単位の評価を説明しましたが、実際には頂点単位の評価を行っています。
視点からの距離や正対する角度、勾配なども加味したリダクションを行っています。

・Houdini標準のポリゴンリダクション機能、Poly Reduce SOPには、
複数視点からの評価機能を備えていますが「影」の判定が無かったので
自作しました。

POLYPHONY™ DIGITAL

補足情報と、まとめ

・あらゆる視点から求められた地形の「稜線」情報は、ポリゴンリダクション以外にも、
シャドウキャスター(リアルタイムシャドウを落とすモデル)の生成や、
オクルージョンカリングを行う為の情報としても活用されました。

・「内積」や「外積」を使用する事で、3次元形状を数学的に評価することができます。
今回、ポリゴンリダクションを例に説明した、操作対象を自動で設定する工程は、
プロシージャルワークフローの構築には必要不可欠な要素です、
しっかりと基本を押さえておくことが大切だと思います。

POLYPHONY™ DIGITAL